NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

USING NEURAL NETWORKS WITHIN THE LEAVES OF A CLASSIFICATION TREE

by

Chen, Chia-sheng

June 2000

Thesis Advisor:	Samuel E. Buttrey
Second Reader:	Lyn Whitaker

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** June 2000	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE
Using Neural Networks Within the Leaves of a Classification Tree

5. FUNDING NUMBERS

6. AUTHOR(S)
Chen, Chia-sheng

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(maximum 200 words)***

Classification trees and neural networks are widely used individually, yet little is known about the effect of combining these two techniques. Earlier work has shown that using *k*-nearest neighbor (*k*-NN) inside the leaves of a tree can increase classification accuracy. Since neural networks are so powerful, we apply neural networks instead of the *k*-NN method inside the leaves of the tree.

This thesis studies the performance of this composite classifier. It is compared to the tree-structured classifier and the neural network classifier. We use commonly available data sets in this application and compare the results to those generated by other generally used classifiers.

Compared to the results of the other two classifiers in this thesis, the composite classifier always gives the lowest cross-validated misclassification error rates in these data sets. Its excellent performance tells us that it is worth further investigation.

14. SUBJECT TERMS
Classification Tree, Neural Networks, nnet.in.leaf method, cross-validation, misclassification rate

15. NUMBER OF PAGES
83

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFI- CATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

THIS PAGE INTENTIONALLY LEFT BLANK

USING NEURAL NETWORKS WITHIN THE LEAVES OF A CLASSIFICATION TREE

Chen, Chia-sheng
Lieutenant Colonel, Army, Taiwan, The Republic of China
B.S., Chinese Military Academy, 1986

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
June 2000

Author: Chen Chia-sheng
Chen, Chia-sheng

Approved by:
Samuel E. Buttrey, Thesis Advisor

Lyn Whitaker, Second Reader

Richard E. Rosenthal
Richard E. Rosenthal, Chairman
Department of Operations Research

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Classification trees and neural networks are widely used individually, yet little is known about the effect of combining these two techniques. Earlier work has shown that using *k*-nearest neighbor (*k*-NN) inside the leaves of a tree can increase classification accuracy. Since neural networks are so powerful, we apply neural networks instead of the *k*-NN method inside the leaves of the tree.

This thesis studies the performance of this composite classifier. It is compared to the tree-structured classifier and the neural network classifier. We use commonly available data sets in this application and compare the results to those generated by other generally used classifiers.

Compared to the results of the other two classifiers in this thesis, composite classifier always gives the lowest cross-validated misclassification error rates in these data sets. Its excellent performance tells us that it is worth further investigation.

THIS PAGE INTENTIONALLY LEFT BLANK

THESIS DISCLAIMER

The reader is cautioned that the computer programs developed in this research may not have been conducted for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

EXCECUTIVE SUMMARY

The classification tree is one of the widely used techniques in classification. Like many other powerful data analytic tools, such as factor analysis and nonmetric scaling, trees were developed in order to cope with actual classification problems based on data.

Like classification trees, the technique of neural networks is also a widely used tool in the literature of statistics. Neural networks, which arise from a variety of sources, have uses ranging from understanding and emulating the human brain, to duplicating human abilities such as speech and the command of language in many disciplines involving pattern recognition, modeling, and prediction (Rohwer, Wynne-Jones, & Wysotzki, 1994). Neural networks are often used as classifiers.

Although classification trees and neural networks are widely used individually, little is known about the effect of combining these two techniques. Earlier work at Naval Postgraduate School has shown that using *k*-nearest neighbor (*k*-NN) inside the leaves of a tree can increase classification accuracy (Karo, 1998). Since neural networks are so powerful, we apply neural networks instead of the *k*-NN method inside the leaves of the tree.

This thesis studies the performance of this composite classifier. We use commonly available data sets in this application and compare the results to those generated by other generally used classifiers.

Compared to the results of the other two classifiers in this thesis, the composite classifier always gives the lowest cross-validated misclassification error rates in the tested data sets. Its excellent performance tells us that it is worth further investigation.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENT

The author wants to thank Professor Buttrey for his guidance and patience during the work. The author is also grateful to Professor Whitaker for her valuable advice.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

1. Classification Tree

The classification tree is one of the widely used techniques in classification. Like many other powerful data analytic tools, such as factor analysis and nonmetric scaling, trees were developed to cope with actual classification problems based on data. Morgan and Sonquist (1963) are the first to work with trees in regression. Breiman and Friedman (1984) use tree methods in classification to deal with actual statistical problems. The problem of classification is the problem of finding a way to assign a new object to one of a number of possible groups. The basic purpose of a classification study, generally speaking, can be either to produce an accurate classifier or to uncover the predictive structure of the problem.

2. Neural Networks

As with classification trees, the technique of neural networks is also one of the widely used tools in the literature of statistics. Neural networks, which arise from a variety of sources, have uses ranging from understanding and emulating the human brain, to duplicating human abilities, such as speech and the command of language in many disciplines involving pattern recognition, modeling, and prediction (Rohwer, Wynne-Jones, & Wysotzki, 1994). Neural networks are often used as classifiers. Fisher (1936) introduces *linear discriminants* as a statistical procedure for classification, from which McCulloch & Pitts (1943) propose the McCulloch-Pitts neuron. In this process, a weighted sum of inputs is acted on by a non-linear function called the activation function. Hebb (1949) notes that if a network of neurons responds in a desirable way to a given input, then the weights should be adjusted to increase the probability of a similar response to similar inputs in the future. Through this work, the functionality of neural networks as determined by the weights of the connections between neurons is finally established.

B. INITIAL CONCEPT

1. Knn-in-leaf Application

The initial concept of this thesis is motivated by a similar application due to Karo (1998). He introduces a new technique, Knn-in-leaf, to the field of classification. Knn-in-leaf is based on Nearest-Neighbor classification. The technique shows an improvement in classification accuracy by about 1 – 3% when k-Nearest Neighbor (k-NN) classification is conducted inside the leaves of a tree. In short, Karo (1998) shows that applying another classifier inside the leaves of the tree produces a "composite" classifier that can have higher accuracy.

2. Neural Networks inside the Leaves of a Classification Tree (Nnet-in-leaf)

Although classification and neural networks are extensively used individually, little is known about the effect of combining these two techniques. Since the nearest neighbor method helps inside the leaves of the tree and since neural networks are so powerful, we would like to apply neural networks instead of the *k*-NN method inside the leaves of the tree.

Our purpose in building this composite is to determine whether it can reduce misclassification rates. We combine the two classifiers in this fashion. A classification tree, produced through binary splits, produces a certain number of terminal nodes or leaves. Each leaf contains a subset of the original data (a "sub-data set"). A neural network classifier is then constructed on each sub-data set separately.

This thesis will measure the behavior of neural networks inside the leaves of a classification tree and will study the performance of this composite classifier. We will use commonly available data sets in this application and compare the results to those generated by other generally used classifiers.

C. PURPOSE OF THIS THESIS

The intention of this thesis is to propose the new Nnet-in-leaf algorithm and study the performance of this composite classifier. This thesis does not have one specific application; the general improvement in classifier accuracy is a topic of interest in many of the applications to the classification trees and neural networks.

D. STRUCTURE OF THIS THESIS

Chapter II discusses the existing tree-structured classifier and the neural network classifier. A graphical example for each classifier is also presented. In the tree-structured classifier section, a self-contained tree algorithm "whole.tree" introduces general tree methods, including how to make a tree, to cross-validate the tree, and to prune it. In the neural network classifier section, a self-contained neural network algorithm "whole.nnet" introduces the neural network methods and prediction methods.

Chapter III introduces the composite classifier made up of the classification tree and the neural network. Its pseudo-code algorithm is provided to enable readers to understand how the Nnet-in-leaf algorithm performs. The description of the data sets used in this thesis is also given in this chapter.

Chapter IV contains results and discussion. The results are given by data set, with each of the baseline and new methods being ranked in their performance. The Measure of Performance (MOP) is simply the misclassification rate, although other aspects of classifier performance are also discussed.

Finally, Chapter V will raise issues for further investigation and present summary conclusions. Two appendices contain raw results and the S-Plus code used to obtain the results.

II. EXISTING TECHNIQUES FOR CLASSIFICATION TREES AND NEURAL NETWORKS

A. CLASSIFICATION TREES

1. Tree-Structured Classifiers

Tree-structured classifiers, which are based on a series of binary decisions, are constructed by repeatedly splitting subsets of the original data set into two descendant subsets. Differences in techniques of the tree construction arise from different choices of splitting rules and pruning rules.

For splitting a leaf, there is a set of features from which to construct splitting attributes. For a binary feature, we will obviously consider only the one possible split on that feature (for example, male versus female). For categorical features with L > 2 levels, we will consider an L-way split, or consider binary splits dividing the levels into two groups (Rohwer R., Wynne-Jones M., & Wysotzki F., 1994).

The second problem is "pruning." For a big data set, the number of rooted subtrees of a binary tree is very large, and there is no good stopping rule. In order not to split "too far," we prune the tree. Breiman *et al.* (1984) introduces the best-known method for tree pruning, "cost-complexity pruning." This pruning method uses the deviance or number of misclassification of a tree penalized by the number of leaves.

In the following section, the method of whole.tree is introduced as a part of the main algorithm the nnet.in.leaf method. The main algorithm will be introduced in the next chapter.

2. Whole.tree Algorithm

The whole.tree algorithm is an extension of the tree-structured classifier. With a view to obtaining a tree's optimal number of terminal nodes and misclassification error rate, this algorithm works sequentially with several tree-related functions. In this thesis we used the S-Plus (1999) statistical package. The S-Plus functions used in this method are tree(), cv.tree(), and prune.tree().

The pseudo-code of the whole.tree algorithm is presented here:

```
Inputs: A given data set (a factor response with two or more levels, plus some continuous
        or factor predictors)
Output: {
Classification tree of the data set:
        Data.tree ← Build a classification tree using the response
        Data.cv ← use cross-validation to find the optimal number of terminal nodes, that
                    is, the number with the lowest deviance
        Data.prune ← prune the Data.tree to the optimal number found in Data.cv
        Leaves ← identify the the sub-data set within each leaf of the optimal tree }
```

3. Example of Classification Tree

A simple example of using the whole.tree method defined in the previous section is given here. The example is based on the wine data, which has 178 observations consisting of three classes, and in which 11 out of 13 independent variables are continuous and the other two are integers. This data has been used and cited (for example, in Kobayashi's *Sensitivity Analysis of the Topology of Classification Trees* (1999)) as an example many times. It is common to justify the appropriateness and usability of an algorithm by using data sets that are familiar in the classification field. A detailed description of this data is given in the next chapter. In Figure 1, the rectangular boxes in the diagram indicate terminal nodes while ellipses indicate non-terminal nodes.

The number inside each terminal node indicates to which class an observation falling into that node is assigned. The numbers under the terminal node indicates the number of observations misclassified and the total number of observations in that node. Figure 1 is a classification tree of the original wine data set.

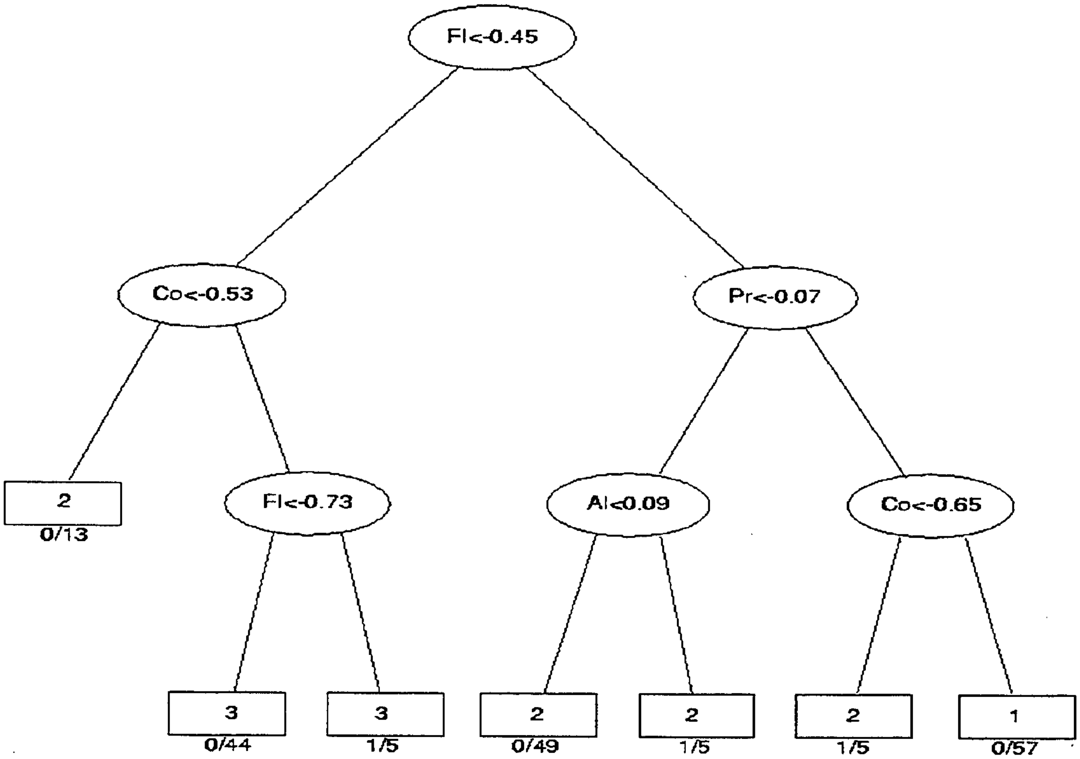


Figure 1: Original Classification Tree from Wine Data Set

Under each terminal node, there are two numbers. The number on the right side represents the total number of observations in that node; the number on the left is the number of observations misclassified. The number inside each terminal node indicates to which class an observation falling into that node is assigned.

For this classification tree, the misclassification error rate is 1.685% and the number of terminal nodes is 7. The misclassification error rate is low: only 3 out of 178 observations are misclassified. However, we can often obtain very low misclassification rates if we produce very large trees. Cross-validation lets us detect this over-fitting. In Figure 2, the cross-validation curve (of tree size versus cross-validated deviance) shows that the numbers of nodes with the lowest deviance are 4 and 5. Therefore, we are convinced that the original classification tree needs to be pruned to guard against overfitting.

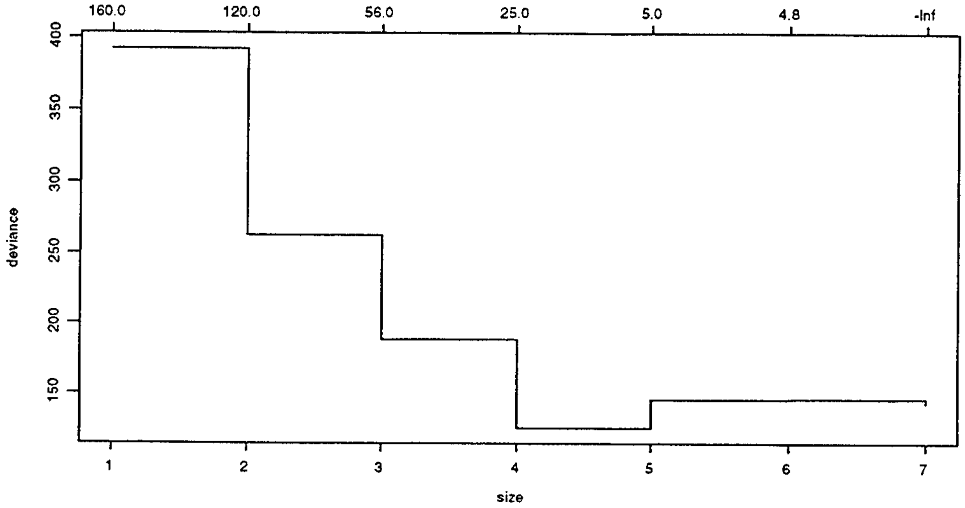


Figure 2: Cross-Validation Plot for the Original Tree

This plot shows the cross-validated deviance as a function of tree size. The tree is pruned to the size for which this is a minimum – here, four nodes.

Figure 3 shows that the pruned tree has 4 terminal nodes with a misclassification error rate 3.37% (6 / 178). The number of the terminal nodes is smaller than that of the original data tree, but the misclassification error rate has gone up from 1.685% to 3.37%. Although this is a good result, our approach is to determine whether the misclassification rates can be reduced.

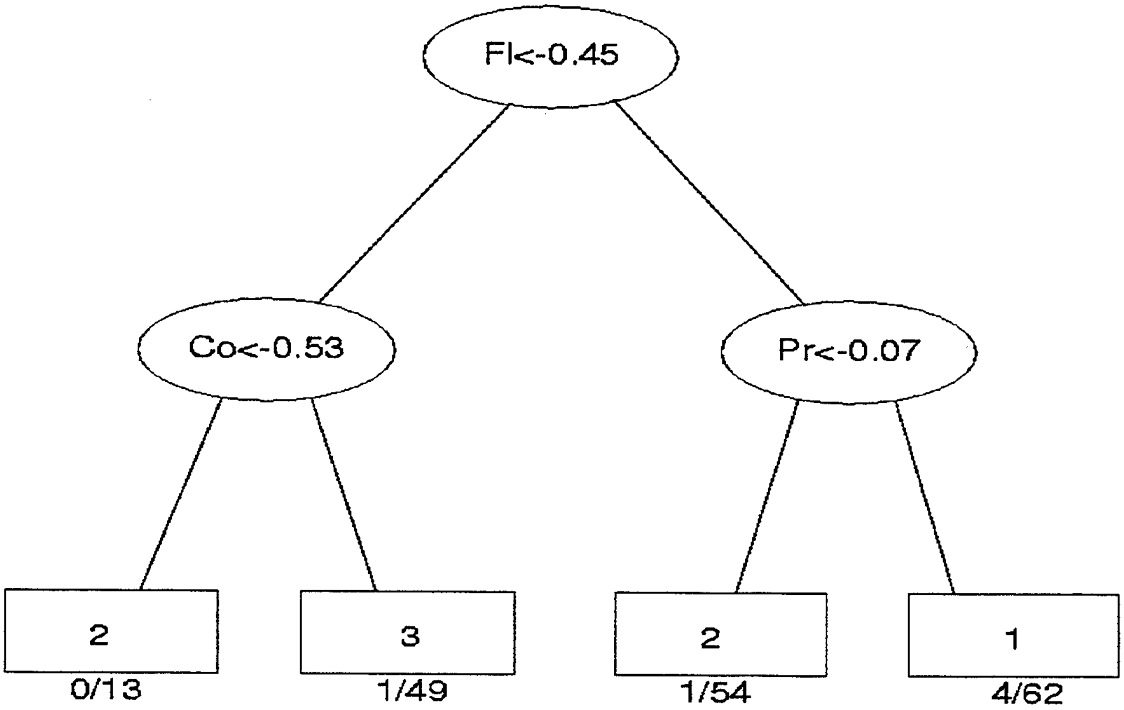


Figure 3: Pruned Classification Tree from Wine Data Set

Based on the same splitting attributes, the pruned tree snipped off the leaves on the "lower" part of the original tree. It creates four new leaves. Because there are some splits in these terminal nodes, the number of misclassified observations increases.

B. NEURAL NETWORKS

1. Neural Network Classifiers

There are many neural network classifiers used in such different fields as machine learning, pattern recognition, and statistical classification. In this thesis, we restrict attention to the most commonly used network for classification, the feed-forward neural network. This classification technique produces a multi-layer network of perceptrons with one hidden layer. This neural network classifier generally has an input layer and an output layer, while the number of hidden layers can be one or more. We consider only the case of one hidden layer.

In neural network classification, generally speaking, there are three parameters governing the algorithm that chooses the weights w_{ij}. The first is the number of nodes in the hidden layer, the second is the weight decay, and the third is the random seed. There is no specific rule for the choice of the number of hidden nodes; therefore, we usually select a number no greater than the number of the attributes of the data set. This results in a reasonable number of weights for the feed-forward calculation; too many hidden units will cause a lot of calculation and may not obtain a better result. For weight decay, the general tendency is that the bigger the weight decay, the faster the convergence. Hinton (1986) reports that weight decay modifies a neural network algorithm to reduce the magnitude of the weights at each step. This only makes sense if the inputs and outputs have been rescaled to the range [0, 1] to be comparable to the outputs of the hidden units. However, the rapid convergence may overlook the minimum point that yields the smallest error rate. The Forensic glass data example conducted by Rohwer R., Wynne-Jones M., & Wysotzki F. (1994) reported this tendency of weight decay. In that example, they use hidden units of two, four, and eight nodes and the weight decays were 0.01, 0.001, and 0.0001. The error rates for two hidden units are 31.8%, 30.4%, and 30.8%. For four hidden units, the error rates are 29.9%, 26.2%, and 23.8%; for eight hidden units they are 29.9%, 26.2%, and 27.1%. The results obviously reinforce the concept just mentioned. As a result, we set the weight decay to $5*10^{-4}$ to try to make sure the algorithm converges. Accompanying weight decay, a random seed gives the neural network classifier a starting point (Ripley, 1996). We study the effect of varying these three parameters in the example of the next section as well as in the next chapter.

2. Whole.nnet Algorithm

The whole.nnet algorithm is a neural network classifier. For the purpose of obtaining a data set's misclassification error rate, this algorithm is designed sequentially with the neural network function and other related functions. The functions used in this method are nnet(), predict(), and table().

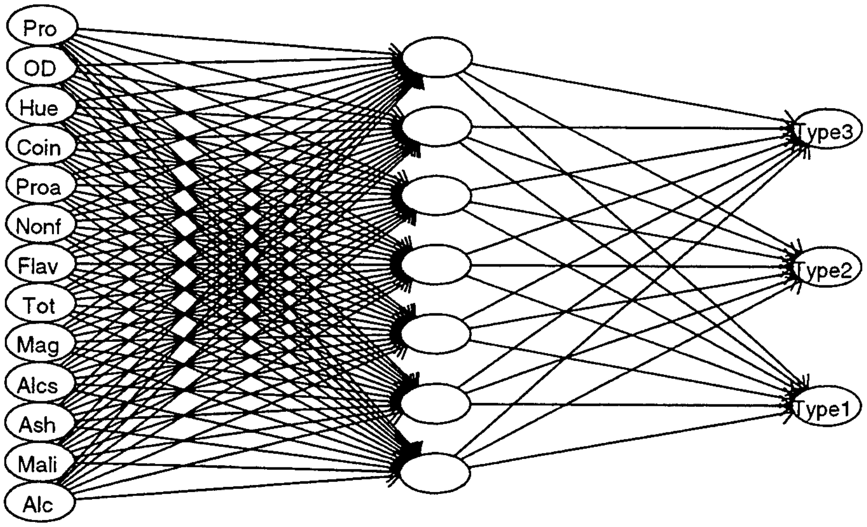
The pseudo-code of the whole.nnet algorithm:

```
Inputs: original: A given data set (a factor response with two or more levels);
                n: number of nodes in the hidden layer
            seed: if supplied, pass to set.seed()
Output: {
Neural networks of the data set :
        Training.set ← sample(data.set)
        Data.nnet ← build neural network on response, using seed, n hidden nodes (The
                      other parameters not mentioned in the function are set to be
                      defaults.)
        Prediction ← Predict test set with the output of the nnet function, Data.nnet.
         Table ← Make a table for the prediction
        Correct.number ← Sum up the number of correct classifications found on the
                    diagonal of this table
        Total.number ← sum up total numbers on the table
        Error.rate ← (Total.number – Correct.number) / Total.number
```

Report the result of the error rate }

3. Example of Neural Networks

In terms of neural network classifiers, a simple example is given here with the whole.nnet method defined in the previous section. The example is based on the sample data we use in the previous section of this chapter, the wine data set, which has 178 observations consisting of three classes, and in which 11 out of 13 independent variables are continuous and the other 2 are integers. The 178 observations are divided into a training and test set of 119 and 59 observations respectively.



```
                              Observed

                   Type 1    Type 2    Type 3
           Type 1     20        0         0
Predicted  Type 2      0       21         1
           Type 3      0        0        17

Giving an error rate of   0.01695
```

Figure 4: Neural Networks and Confusion Table from Wine Data Set

The matrix is a confusion table, based on the network. The numbers on the diagonal represent the number of correct classifications. Numbers other than the numbers on the diagonal are the numbers of misclassification errors. The error rate of the wine data set using this whole.nnet method is based on the 7 hidden nodes and a seed of 100.

The error rate we obtain from the whole.nnet method is 1.695%, which is quite close to the error rate of 1.685% from whole.tree method. In Figure 4 the 13 nodes in the input layer represent the 13 independent variables of the wine data set. In the middle of this figure is the hidden layer of seven nodes, a number supplied by us. The output layer has three nodes representing the three classes in this data to specify the neural network classification. In this example, the number of observations of the training set is two-thirds of the original wine.1 data set, and the test set is one-third. The report tells us that the error rate is one out of 59. The number, 59, is the sum of the numbers on the diagonal of the confusion table, representing the total number of the test set. The number, one, is the number of misclassification error, which is not on the diagonal of the confuse table.

In order to see the variation on the error rates caused by the number of nodes in the hidden layer, we set the number to 13, the same as the number of the independent variables of the wine data, with the same random seed 100. The error rate increases to 5.1%. When we permit only 2 nodes in the hidden layer and maintain the random seed of 100, the result of the error rate is even higher than that with 13 nodes in the hidden layer.

We vary the number of nodes in the hidden layer, from 1 to 30 (see Appendix C), with a fixed random number seed. The error rates range from1.7% to 11.86% with 17 out of 30 giving the minimum error rate of 1.7%.

When we apply the neural network classifier within the leaves of a classification tree, we hope, of course, to see the minimum error rate instead of the average error rate. Based on this idea, we use a loop over the number of nodes in the hidden layer as well as looping over a set of random seeds in the nnet.in.leaf method to be discussed in the next chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

III. DEFINITION OF NEURAL NETWORKS INSIDE THE LEAVES OF A CLASSIFICATION TREE (NNET.IN.LEAF)

A. PROPOSED NEW CLASSIFICATION RULE

The combination of classification trees and neural networks is a new direction in this literature, although both have been commonly used in the field of classification for a long time. The tree-structured classifier is a very successful classification application, as are neural networks. The goal of this composite classifier is to create a more reliable classification method.

1. Definition

A nn.in.leaf method is an algorithm in which a neural network classifier is used within the leaves of a classification tree. When a data set is classified with a tree-structured classifier, the result is a set of terminal nodes, a sub-data set within each terminal node, and the overall misclassification error rate. The tree-structured classifier error rate measures how pure the leaves are. To make the leaves more accurate, the neural network classifier then operates on the data within each leaf. The overall error rate from all neural networks is the error rate of this nnet.in.leaf method.

2. Preprocessing Concerns

Before a data set is classified with this method, some preprocessing is necessary. With the use of S-Plus software and the format of this nnet.in.leaf algorithm, the following has to be done prior to classification with our software.

a. Arbitrary Position of Response of a Data Set

Each data set has its own particular format. Because of the way the algorithm is coded and run, the response is required to be the first column of the data.

b. Random Seed and Number of Nodes in the Hidden Layer

In neural networks, random seeds and number of nodes in the hidden layer are two important inputs. Different combinations of these two inputs result in different error rates. Therefore, proper selection of these two inputs is time-consuming and demanding.

The nnet algorithm uses a random seed to select starting values for the weights. The user must also specify the number of nodes in the hidden layer (the number is also called "network size"). This number is usually taken to be less than the number of levels of the response variable. In the nnet.in.leaf algorithm, the number of levels of the response in sub-data sets is never larger than that in the original data set. The algorithm tries every combination from a set of network sizes and a set of random seeds; when these sets are large, the algorithm will run slowly or crash.

c. Factor Response and Its Levels

This algorithm deals only with data sets which have factor responses. If the factor response has two levels, the nnet method of Ripley (1999) produces one output and entropy fit, and a number of outputs equal to the number of classes and a softmax output stage for more levels.

3. Cross-Validation

In the nnet.in.leaf algorithm, two phases of cross-validation have been used to evaluate the method. These are discussed in detail so that the terminology as it is used later is clear. Phase one is performed within the classification tree. It cross-validates the tree sequence obtained by pruning a tree by partitioning the original data set into a number of distinct sub-data sets, fitting subtree sequences to these, and using a sub-data set previously held out to evaluate the sequence. In other words, at the beginning for a data set running through this algorithm, a tree is created with a number of terminal nodes or leaves. The number of terminal nodes may or may not be the optimal number that gives the lowest deviance. The function of the cross-validation is to find this lowest deviance so as to find the tree with an optimal number of terminal nodes; the tree is then pruned to this size.

Phase two is conducted within the neural networks part of the nnet.in.leaf algorithm. When the optimal number of terminal nodes and the sub-data sets within each node are obtained, each sub-data set is broken randomly into a training set and a test set. After the output of the nnet method in the nnet.in.leaf is obtained, the next step is a prediction of the test set and a computation of the error rate.

4. Pseudo-Code of Algorithm

A basic pseudo-code of the algorithm to find sub-data sets within each terminal node and to find the overall misclassification rate is as follows:

```
Inputs:
        A given data set (a factor response with two or more levels) : data.set
        A set of numbers of nodes in the hidden layer : Ni
        A set of random seeds for Neural networks : Sj
Output: {
Classification tree of the data set :
        Data.tree ← build a classification tree
Data.cv ← use cross-validation to find the optimal number of terminal nodes with
        the lowest deviance
Data.prune ← prune the Data.tree with the optimal number found in the Data.cv
Leaves ← identify the the sub-data set within each leaf of the tree with optimal
        number of terminal nodes
Sub.leaf ← the sub-data set within a leaf of the tree
For each leaf {
  Scale Sub.leaf so that each prediction has mean 0 and standard deviation 1
  For each number of nodes in the hidden layer (Ni) {
    TotalCount ← set to zero
    ErrorCount ← set to zero
    TotalErrorRate ← set to zero
    Training.set ← sample(sub.leaf) // Random sample of Sub.leaf, each
                  observation having probability 0.67 of being on the sample
    Nnet.output ← build neural network on Training.set
    For each value of random seed (Sj) {
      Pred ← predict the test set (using Nnet.output)
      TotalCount ← TotalCount + number of test set observations
      ErrorCount ← ErrorCount + number of test set errors
    }// End for a single leaf with a single number of nodes in hidden layer
  }// End for a single leaf
  TotalErrorRate ← ErrorCount / TotalCount
} // End for the calculation of the nnet function
```

```
    return TotalErrorRate, the misclassification error rate of nnet.in.leaf
}
```

B. TEST METHODOLGY

1. Assumptions

The major assumption made for the nnet.in.leaf methodology in this thesis is that the efficiency of the method is not to be considered. That is, the only Measure of Effectiveness for the algorithm is its ability to improve the accuracy of the classifications. It is assumed that if the method has the potential for further investigation then its application to data sets with numeric responses can be developed.

2. Data

The data sets used for this algorithm are taken from the UC Irvine Machine Learning Repository (Merz and Murphy, 1996). For simplicity the data sets are selected according to the following criteria:

1. Factor response for this algorithm
2. No missing data
3. A mid-sized to large data set, in the range of 150-20,000 items

The data sets which are used in this thesis are described below. To distinguish the original data sets from those used in this algorithm, those data sets which have ".1" attached to the end of that single word name in bold at the start of each paragraph are reformatted by relocating their factor responses on the first column in the data frame. Each set is described in sufficient detail to understand the purpose of the classification. Where other results (i.e. misclassification rates reported in UCIrvine Machine Learning

Repository as well as those from re-trails of classification trees and neural networks in this thesis) are known, they are also given here.

A summary of the data sets used is given in the following table:

Data name		Classes	Cases	Attributes
a	iris.1	3	150	4
b	wine.1	3	178	13
c	glass.1	6	214	9
d	vowel.1	11	990	11
e	letter.1	26	20000	16
f	sonar.1	2	208	60
g	diabetes.1	2	768	8

Table 1: A Summary of Data Sets with Respective Numbers of Classes, Cases and Attributes

a. iris.1

This data set is a copy of the original data set ***iris***, made famous by Fisher (1936), and is built into S-Plus. The data set consists of measurements on 150 flowers, 50 from each of 3 iris species *Setosa*, *Versicolor*, and *Virginica*. The four continuous attributes are sepal length and width, and petal length and width.

b. wine.1

This wine.1 data set duplicates the ***wine recognition*** data set, which comes from a chemical analysis of wines grown in a particular region in Italy. The data consist of 178 cases: 59 of class-1 wines, 71 of class-2 wines and 48 of class-3 wines. Each case is composed of thirteen continuous attributes measuring the chemical properties of the wines. They are 1) Alcohol, 2) Malic acid, 3) Ash, 4) Alkalinity of ash, 5) Magnesium, 6) Total phenols, 7) Flavanoids, 8) Nonflavanoid phenols, 9) Proanthocyanins, 10) Color intensity, 11) Hue, 12) OD280/OD315 of diluted wines and 13) Proline. The data are downloaded from the UC Irvine Data Repository.

c. glass.1

The glass.1 data set is a copy of the original ***glass*** data set, which is downloaded from the UC Irvine Data Repository. The data consist of 214 cases containing seven classes and nine continuous attributes. The class distribution is 70 float-processed building window glasses, 17 float-processed vehicle window glasses, 76 non-float-processed building window glasses, zero non-float-processed vehicle windows, 13 containers, 9 tableware items and 29 headlamps. All attributes are continuous. They are Refractive index, Sodium, Magnesium, Aluminum, Silicon, Potassium, Calcium, Barium, and Iron.

d. vowel.1

The vowel.1 data set is a copy of the original ***vowel recognition*** data set, which is also downloaded from the UC Irvine Data Repository. The data consist of 990 cases containing eleven classes and eleven integer attributes. This data set could be seen as a three dimensional array: speaker, vowel, and input The speakers are indexed by integers 0-89. (Actually, there are fifteen individual speakers, each saying each vowel six times.) The vowels are indexed by integers 0-10 referring to sounds labeled "i", "I", "E", "A", "a:", "Y", "O", "c:", "U", "u", and "3:". For each utterance, there are ten floating-point input values, with array indices 0-9.

e. letter.1

This letter.1 data set duplicates the ***letter recognition*** data set from the UC Irvine Data Repository. The objective is to identify each of a large number of black-

and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. The character images were based on 20 different fonts and each letter within these 20 fonts was randomly distorted to produce a file of 20,000 unique stimuli. Each stimulus was converted into 16 primitive numerical attributes (statistical moments and edge counts, like, for example, the average of x^2y) which were then scaled to fit into a range of integer values from 0 through 15.

f. Sonar.1

This is the data set used by Gorman and Sejnowski (1988) in their study of the classification of sonar signals using a neural network. The task is to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. The data set contains 208 cases of which 111 cases are obtained by bouncing sonar signals off a metal cylinder at various angles and under various conditions, and of which 97 cases are obtained by bouncing signals off of roughly cylindrical rocks under similar conditions. The transmitted sonar signal is a frequency-modulated chirp, rising in frequency. The data set contains signals obtained from a variety of different aspect angles, spanning 90 degrees for the cylinder and 180 degrees for the rock. There are two classes, and each case is a set of 60 numbers in the range 0.0 to 1.0. This data set is also downloaded from the UC Irvine Data Repository.

g. Diabetes.1

The data set of diabetes.1 is a copy of the Pima Indians Diabetes Database. This database is developed by the National Institute of Diabetes and Digestive and Kidney Diseases and downloaded from the UC Irvine Data Repository. The data contains measurements of 8 attributes from Pima Indian (near Phoenix, Arizona) women aged over 21 and a classification, 2 classes, as to whether or not they have diabetes. The data set contains 768 observations of which 500 are class 0, no diabetes, and 268 cases are class 1, having diabetes. The eight attributes are numeric and are as follows:

- number of pregnancies
- plasma glucose concentration after a two-hour oral glucose tolerance test
- diastolic blood pressure
- triceps skin fold thickness
- two-hour serum insulin
- body mass index (weight in kg/(height in m)^2)
- diabetes pedigree function
- age

This data set has been well studied, with the first reported use being by Smith, et al. (1988) using their ADAP routine.

3. S-Plus Code and Functions

All algorithms were coded in the S Version 3 language, in S-Plus 2000. S-PLUS 2000 is a major upgrade of S-PLUS, built on the core S Version 3 language from Lucent Technologies used in S-PLUS 4.5 and earlier releases of S-PLUS for Windows with classification functions written by Venables and Ripley (1997).

THIS PAGE INTENTIONALLY LEFT BLANK

IV. RESULTS AND DISCUSSION

A. PRELIMINARIES

This chapter will present results and discussion for each of the data sets examined in the nnet.in.leaf method along with the other two classification methods, whole.tree and whole.nnet. The whole.tree and whole.nnet methods produce the baseline misclassification rates. (More detailed results are in Appendix B). Where other results for these data sets are available from the literature, these will be used to see how the nnet.in.leaf method performs compared to the results of other investigations. All results here will be expressed as misclassification percentages with the raw number misclassified and other data set details such as size in the appendix. Also Appendix B contains details on the optimal size of the trees used, the random seeds picked, the threshold of the prediction, the number of nodes in the hidden layer, the weight decay and scaling used, and so on.

1. Data Set Types

In order to make the distinction of the names of the data sets used in this thesis from their original names, those data sets used in this thesis have been renamed by adding ".1" to the end of the name. These date sets differ from the original only in that the response is moved to the first column. Data set sonar.1 and diabetes.1 each have two classes. The rest have more; for example, iris.1 has 3 levels and letter.1 has 26 levels.

Finally, it should be noted that some data sets respond well to scaling of the variables. By standardizing each variable to have a mean of 0 and a standard deviation of 1 the influences of individual variables are equalized. In some data sets scaleing made no difference. Results are also reported for each data set with the data scaled.

B. RESULTS

Misclassification error rates are given with the reports of each data set below. Each data report consists of three parts: Classification tree error rates (from whole.tree), Neural network misclassification error rates (from whole.nnet), and nnet.in.leaf misclassification error rates. Detailed results are shown in the Appendix B. Prior to an inspection of the results of the main method of nnet.in.leaf algorithm, two important points need to be stated. First of all is the stability of the misclassification error rate of tree-structured classifiers. Generally, tree-structured classifiers work quite well for most of the data sets. They often give consistent results in the way of binary splitting. But for some data sets, one with a 2-class response and one with a response having more than 40 classes, the tree-structured classifiers perform unpredictably. We will see these results later. To find out the usability of a classifier, the data set is usually split into two parts, a training set and a test set, and examined by the method of whole.tree.

The second benchmark is a neural network classifier. The whole.nnet method uses a training set and a test set for cross-validation and prediction. The misclassification error rates reported from the neural network classifier are generally smaller than those from tree-structured classifiers, when the number of nodes in the hidden layer and the number of random seeds are appropriately picked.

In the nnet.in.leaf method, the cross-validation by using three for-loops is intended to find the minimum error rates. These three for-loops are over the number of optimal leaves, a vector of numbers of nodes in the hidden layer, and a vector of random seeds.

1. IRIS.1 DATA

Figure 5, the cross-validation plot of the iris.1 classification tree, shows the different values of deviance for each number of terminal nodes. The lowest points on the curve (corresponding to four and five leaves) are the optimal sizes of the trees; either one of them could be chosen as the "best" number of terminal nodes. The misclassification rate with the pruned tree is 2.67%.

The results of the whole.tree, whole.nnet and nnet.in.leaf methods are shown in Table 3. For the iris.1 data, both the neural network classifier and the nnet.in.leaf algorithm have good performance on the classification. The misclassification rates were 0%. Scaling does not affect the results. Since the whole.nnet method shared the same technique for this data set with the nnet.in.leaf method, the nnet.in.leaf also gives a good result, with an error rate of 0%. The ranking of the three methods is also shown in Table 3, which indicates that the nnet.in.leaf method and whole.nnet method are better than the whole.tree method. In the UC Irvine database, the results collected from previous work show that very low misclassification rates for this data are obtained and reported in many publications. Dasarathy (1980) reports an error rate of 0% for the setosa class and an overall error rate of 2.67%.

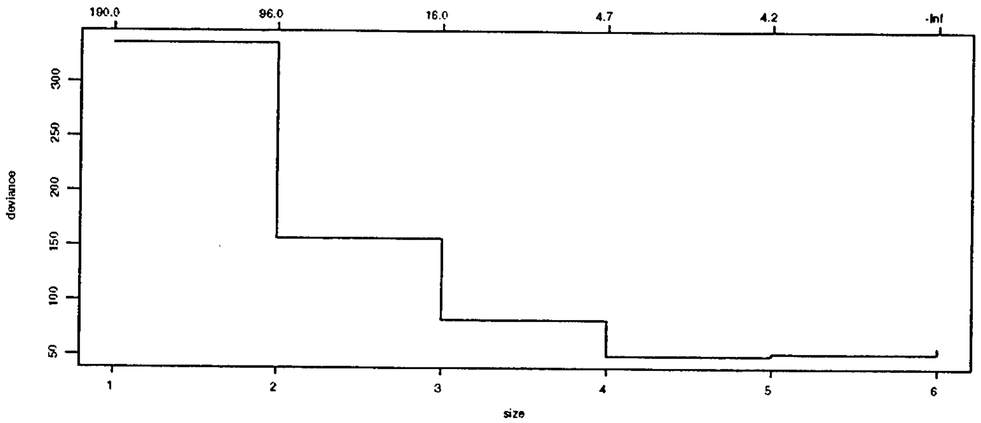


Figure 5: Cross-Validation of iris.1 Classification Tree

Data set : Iris.1	Classifiers:	Whole.tree misclassification error rate	Whole.nnet misclassification error rate	Nnet.in.leaf misclassification error rate
Unscaled	Misclassification rate	2.67%	0%	0%
	Rank	2	1	1
Scaled	Misclassification rate	2.67%	0%	0%
	Rank	2	1	1

Table 2: Misclassification Error Rates for iris.1 Data Set

2. WINE.1 DATA

Figure 6, the cross-validation plot of the wine.1 classification tree, shows that the optimal size is 7. We use the optimal number to prune the tree. After finding the optimal number of terminal nodes, we examine the results of the nnet.in.leaf method and the whole.nnet method. The results shown in Table 3 indicate that there is a large improvement for nnet.in.leaf, but not for whole.nnet. The whole.nnet method gives a misclassification error rate for the wine.1 data set of about 1.7%.

On the other hand, the nnet.in.leaf method produces a misclassification rate of 0%. The scaling of the wine.1 data set does not affect the results. In the UC Irvine database, the misclassification rate attached by Aeberhard, Coomans and de Vel (1992) is reported to be 0%. The report indicates that the classes are separable, and achieves 100% correct classification. This error rate is determined by using the leave-one-out technique.

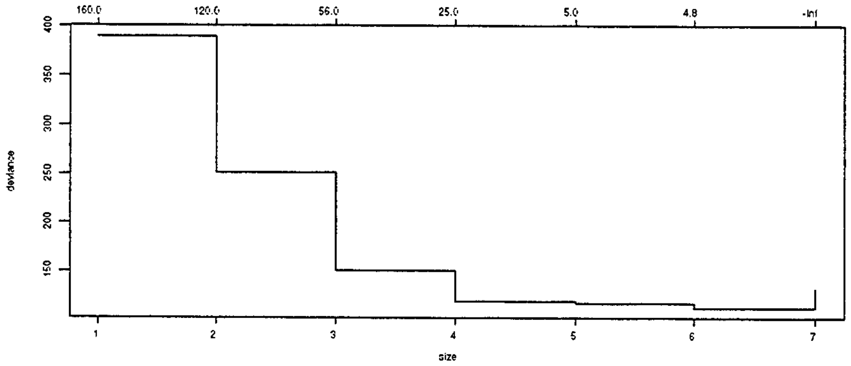


Figure 6: Cross-Validation of wine.1 Classification Tree

Data set : wine.1	Classifiers:	Whole.tree misclassification error rate	Whole.nnet misclassification error rate	Nnet.in.leaf misclassification error rate
Unscaled	Misclassification rate	1.69%	1.7%	0%
	Rank	2	3	1
Scaled	Misclassification rate	1.69%	1.7%	0%
	Rank	2	3	1

Table 3: Misclassification Error Rates for wine.1 Data Set

3. GLASS.1 DATA

Applying the tree-structured classifier to the glass.1 data gives very interesting results. The error rate of 0% clearly shows that the tree-structured classifier correctly classifies all of the data. The cross-validation of the data tree shown on the Figure 7 went all the way down to the lower right corner with the optimal terminal node size of 6.

The misclassification error rates in Table 4 presented a comparison on the three classifiers. The nnet.in.leaf algorithm benefited from the combination of the tree-structured classifier and the neural network classifier. The tree method gave a good result for the glass.1 data set, and so did the nnet.in.leaf method. All misclassification rates were 0%. For the whole.tree method, the performance of the whole.nnet method and the nnet.in.leaf method had the same error rate. No misclassification rates for the glass.1 data were available in the UC Irvine database.

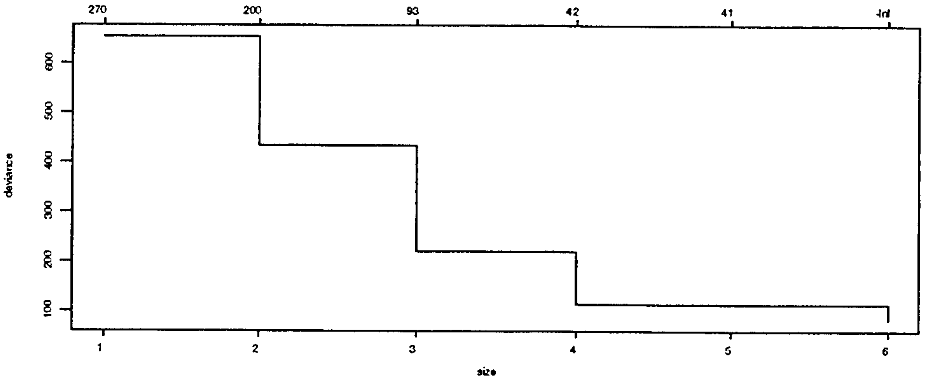


Figure 7: Cross-Validation of glass.1 Classification Tree

Data set: glass.1	Classifiers:	Whole.tree misclassification error rate	Whole.nnet misclassification error rate	Nnet.in.leaf misclassification error rate
Unscaled	Misclassification rate	0%	0%	0%
	Rank	1	1	1
Scaled	Misclassification rate	0%	0%	0%
	Rank	1	1	1

Table 4: Misclassification Error Rates for glass.1 Data Set

4. VOWEL.1 DATA

In Figure 8, the optimal number of the terminal nodes for the vowel.1 data found by cross-validating the classification tree is 43, which is large for a classification tree. A classification tree with a large number of terminal nodes and a 22% misclassification rate based on a data set with 11 classes does not carry much information.

For the nnet.in.leaf method and the whole.nnet method, the results shown in Table 5 are more impressive. Compared with the whole.tree method, the results of the misclassification rate for the whole.nnet method decrease to 13.5%. This improvement is significant. But the improvement made by the nnet.in.leaf method is even more inspiring. The misclassification rate decreases from 22% to 1.4% when the data are unscaled, and decreases to 0.9% for scaled data. In the UC Irvine database, the best misclassification rate for this data is about 34%, reported by Robinson (1989). He uses 16 different classifiers, including the nearest neighbor and neural networks, but due to the computational limits, the result was no better than 66% classification accuracy.

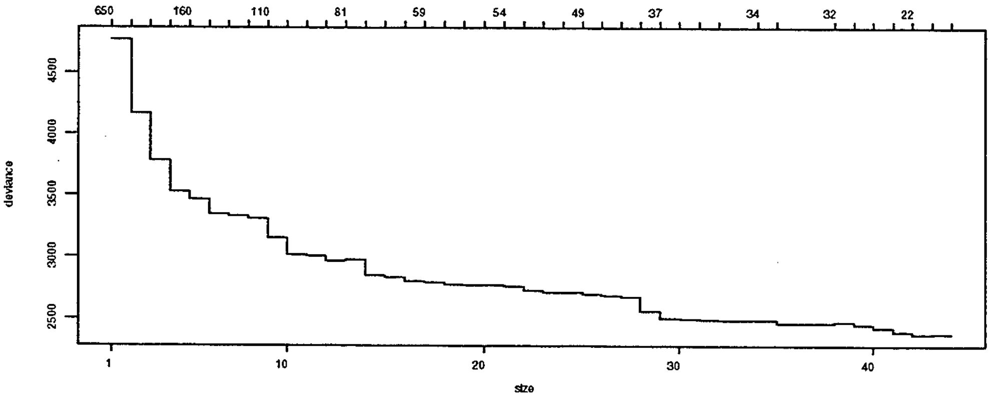


Figure 8: Cross-Validation of vowel.1 Classification Tree

Data set : vowel.1	Classifiers:	Whole.tree misclassification error rate	Whole.nnet misclassification error rate	Nnet.in.leaf misclassification error rate
Unscaled	Misclassification rate	21.21%	13.5%	1.4%
	Rank	3	2	1
Scaled	Misclassification rate	21.21%	13.5%	0.9%
	Rank	3	2	1

Table 5: Misclassification Error Rates for vowel.1 Data Set

5. LETTER.1 DATA

The results of the data set letter.1 are a little different from those of the other data sets. First, this data set is large, having 20,000 observations and 26 classes. It is not easy for tree-structured classifiers to classify such a large data set and to give small misclassification rates. Therefore, the error rate of this classification tree, about 38.3%, was not very surprising. For the classification tree in this data set, a 38.3% misclassification error rate means 7660 out of 20000 observations are misclassified. The cross-validation of the data tree shown on the Figure 9 goes all the way down to the lower right corner with the optimal number of terminal nodes equal to 66.

In Table 6, the misclassification rate is 5.5% for the nnet.in.leaf method and 21% for the whole.nnet method. The reduction of the error rate from 38.3% to 5.5% strongly indicates that the improvement of classification with the nnet.in.leaf method is significant. The whole.nnet method itself also has an improvement when compared to the whole.tree method. We encounter some difficulty in using the nnet.in.leaf method on this data due to the size of the data set and the large number of terminal nodes. Although the nnet.in.leaf method gives significant improvement in classification, it is time-consuming. For this particular data set, the very low misclassification error rate can not be achieved by either tree-structured classifiers or neural network classifiers alone. In the UC Irvine database, P. W. Frey and D. J. Slate (1991) report that the lowest achieved misclassification error rate is about 20%. Their research for the letter.1 data set uses several variations of Holland-style adaptive classifier systems to learn to guess the letter categories correctly. This rate is smaller than that of the tree-structured classifiers, and about the same as the error rate with the neural network classifier in this thesis. However, the nnet.in.leaf method has a much better misclassification rate than this 20% error rate.

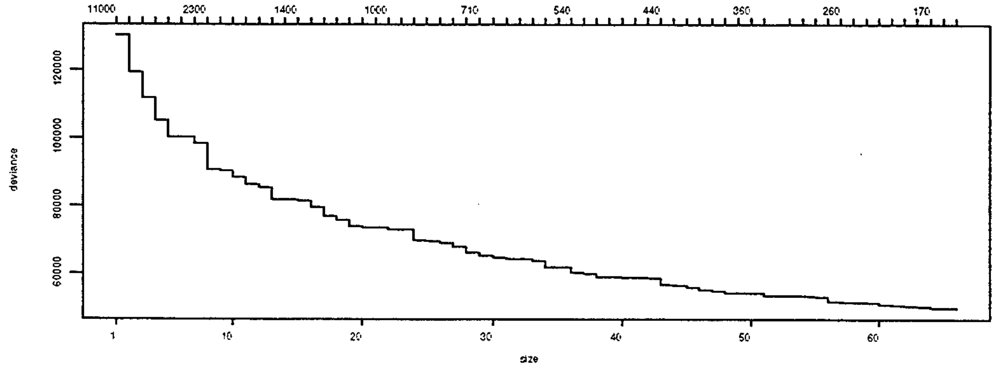


Figure 9: Cross-Validation of letter.1 Classification Tree

Data set : letter.1	Classifiers:	Whole.tree misclassification error rate	Whole.nnet misclassification error rate	Nnet.in.leaf misclassification error rate
Unscaled	Misclassification rate	38.29%	21%	5.5%
	Rank	3	2	1
Scaled	Misclassification rate	38.29%	21%	5.5%
	Rank	3	2	1

Table 6: Misclassification Error Rates for letter.1 Data Set

6. SONAR.1 DATA

In Figure 10, the curve shows that the optimal number of terminal nodes with the lowest deviance are 2, 3, 4, and 5. With the cross-validation of the sonar.1 data tree, the 2 terminal nodes are better for the nnet.in.leaf method. The plot behaves differently because the response of the data set has 2 classes.

In Table 7, the result of the whole.nnet method shows an improvement. The misclassification error rate is reduced to 15.9%, compared to the classification tree error rate of 24.04%. The result of the nnet.in.leaf method is even better; the misclassification error rate decreases to 10.1%. In the UC Irvine database, the best result is a misclassification error rate of 17%, which is about the same as the error rate that Karo (1998) achieves using the *k*-NN classifier. (His knn.in.leaf, scaling the data set and using leave-one-out cross-validation, obtains a misclassification rate of 11.1%.)

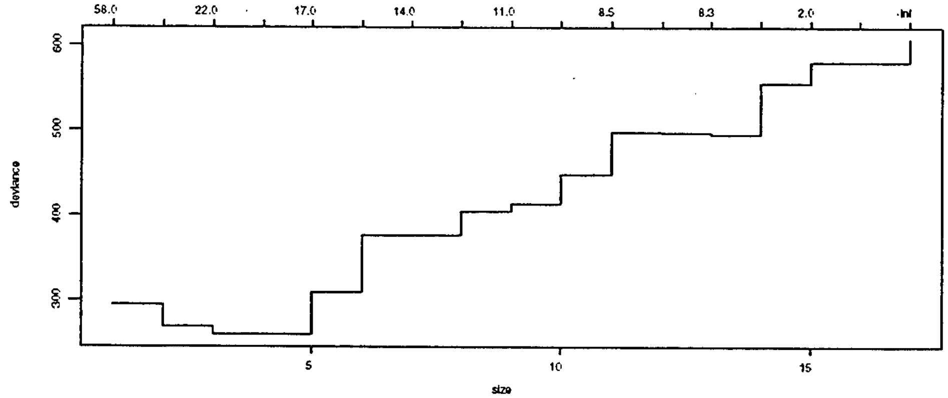


Figure 10: Cross-Validation of sonar.1 Classification Tree

Data set : sonar.1	Classifiers:	Whole.tree misclassification error rate	Whole.nnet misclassification error rate	Nnet.in.leaf misclassification error rate
Unscaled	Misclassification rate	24.04%	15.9%	10.1%
	Rank	3	2	1
Scaled	Misclassification rate	24.04%	15.9%	10.1%
	Rank	3	2	1

Table 7: Misclassification Error Rates for sonar.1 Data Set

7. DIABETES.1 DATA

Figure 11 shows clearly that the optimal number of the terminal nodes is between 7 and 15. On the other hand, the lower error rate of about 11.07%, achieved with the 71 leaves, has a very high cross-validated deviance. Cross-validation of the diabetes.1 data shows that the optimal size of 7 is better for the nnet.in.leaf method.

Table 8 shows the improvements made by the whole.nnet method and the nnet.in.leaf method, compared to the whole.tree method. The neural network classifier reduces the misclassification rate to 16.1%, compared to the classification tree error rate of 22.79%. The nnet.in.leaf method improves the error rate by decreasing it to 14.5% with unscaled data and all the way to 12.5% with scaled data. The best reported result for this data set in the Statlog project is 22.3% in 12-fold cross-validation.

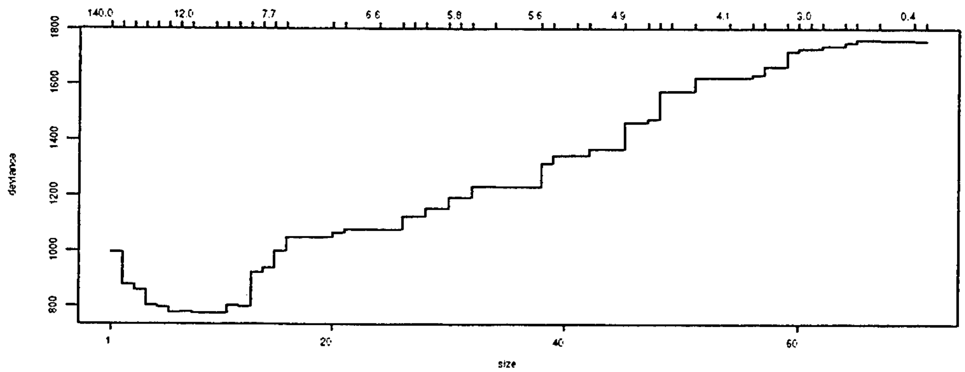


Figure 11: Cross-Validation of diabetes.1 Classification Tree

Data set : diabetes.1	Classifiers:	Whole.tree misclassification error rate	Whole.nnet misclassification error rate	Nnet.in.leaf misclassification error rate
Unscaled	Misclassification rate	22.79%	16.1%	14.5%
	Rank	3	2	1
Scaled	Misclassification rate	22.79%	16.1%	12.5%
	Rank	3	2	1

Table 8: Misclassification Error Rates for diabetes.1 Data Set

V. CONCLUSIONS AND FURTHER RESEARCH

A. CONCLUSIONS

In this thesis we propose the nnet.in.leaf method and compare its misclassification rates with those of two other classifiers. We also examine the performance of this proposed method. It is apparent that the nnet.in.leaf method is a viable first approach to many classification problems, and it is also worth further investigation. Compared to tree-structured classifiers and neural network classifiers, the performance of the nnet.in.leaf method has some merits.

First, the nnet.in.leaf method always gives the lowest misclassification error rates in our data sets. Table 9 shows that its performance always ranks first among the three methods. Scaling of the data does not influence this ranking.

Second, this method is also capable of dealing with a large data set, such as the letter data and the vowel data, which were used in the previous chapter. The letter.1 data has 20,000 observations with 26 classes. The large number of observations makes some common algorithms, like knn-in-leaf (mid-sized data, ranging from 200 to 1000 items), less capable of dealing with it. However, the nnet.in.leaf method is not only able to classify this data, but also gives a very encouraging misclassification error rate.

For the tree-structured classifiers and neural network classifiers, although we cannot conclude that the latter generally does better than the former, we are confident that the performance of the neural networks in these five out of seven data sets is better than the tree's performance.

However, the nnet.in.leaf method also has limits and weakness. It is time-consuming to run a data set with more than 50 classes (like the AF.1 data set, see

Appendix C). In addition, use of weight decay and scaling to decrease the influence of the random seed should make the nnet.in.leaf method be more stable. However, the results of these eight data sets have shown that they made no difference.

Data set :	Scaling	Whole.tree result's Ranking	Whole.nnet result's Ranking	Nnet.in.leaf result's Ranking
iris.1	Unscaled	2	1	1
	Scaled	2	1	1
wine.1	Unscaled	2	3	1
	Scaled	2	3	1
glass.1	Unscaled	1	1	1
	Scaled	1	1	1
vowel.1	Unscaled	3	2	1
	Scaled	3	2	1
letter.1	Unscaled	3	2	1
	Scaled	3	2	1
sonar.1	Unscaled	3	2	1
	Scaled	3	2	1
diabetes.1	Unscaled	3	2	1
	Scaled	3	2	1

Table 9: Ranking of These Three Classifiers

B. FURTHER RESEARCH

We propose a composite classifier consisting of a classification tree and neural networks. Since classification trees have been widely used in fields like medical diagnostics and botany for many years, and neural networks have been widely used in pattern recognition for a few decades, their combination is promising for classification in those fields. Further study of the composite estimation applied to data from these fields needs to be done.

Because of its excellent performance, the nnet.in.leaf method might also be good for the classification of the assignments of military personnel. Military assignments are based on the personnel's service, branch, rank, specialty, age, military performance, education, sex, and so forth. (may be even including height, weight and so on). Those attributes are very similar to those of the diabetes.1 data set we examined in chapter four.

Although this method performs well, it still needs to be improved. If the weight decay and scaling could be adjusted so as to produce stable misclassification rates, this method would be more reliable.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. S-PLUS CODE

This appendix contains the S-Plus code for functions used to test the classification methods in this thesis. The heading for each function is its name, and the first comment block provides a description of the purpose of the function. Other functions used included standard S-Plus functions for classification trees (S-Plus, 2000), along with a library of classification functions produced by Venables and Ripley (1997). These functions are also described below.

A. Nnet FUNCTIONS

1. nnet

This function is a standard *neural network* algorithm for classifying a test set against a known training set. It was part of a library of classification functions written by Venables and Ripley. The function is used together with the predict() function to return the classification of the test set instances. This function is widely used for pattern recognition.

2. predict

This function is from the Venables and Ripley library. It performs a test set on a fitted model object from a training set. The function returns the predicted classifications of the test set. It is heavily used in other functions.

B. nnet.in.leaf Algorithm

nnet.in.leaf

```
function(original, n = 1:50, seed = 120, scale = T, threshold = 0.5, decay = 5e-4,
        use.new.version = T)
{
#
# By Sam Buttrey and Chia-sheng Chen
#
# nnet.in.leaf: The overall strategy consists of two stages. Stage one: Classification tree.
#First of all, build a classification tree for the original data set. After that, with the
#classification tree, find out the optimal number of terminal nodes and the sub-data sets
#within each node by using cross-validation. Then, as soon as the optimal number of
#terminal nodes is found, prune the tree to the optimal number of nodes.
#Stage two: Neural networks. To begin with, scale the sub-data sets within each leaf.
#Perform neural networks with each sub-data set. Then, predict and sum up the errors.
#Finally, give the error rates of this nnet.in.leaf algorithm.
#
# Arguments:
#              original: Full set of data
#                     n : Number of nodes in hidden layer
#                  seed : Random seed: if supplied, pass to set.seed()
#                  scale: If true, scale each sub-data set.
#                 decay: Set to non-zero value in the hoping of getting consistent results
#   use.new.version: If "use.new.version" is TRUE, we use nnet.formula.new, which
#                        treats two-level factor responses the same as multi-level ones.
#           threshold: Provided for prediction, usually set to be 0.5 for 2-class response.
#                        For more than 2-class response, we use max.col() to select the
#                        column which has the maximum value in it.
#
#  return value : 1.) optimal number of terminal nodes in classification tree
#                 2.) misclassification error rate in classification tree
#                 3.) misclassification error rate in nnet.in.leaf
#
#
#
# For nnet.in.leaf
# We want to divide this algorithm into several steps:
# Step 1: Call .First() function from S-Plus Library to set nnet function ready to use. We
#define some variables for future use. Define variable opt.treesize for the result of cross
#validating the tree we get from the original data set, and define variables total.error.num
#and total.error.denom as the total number of errors of the numerator and denominator.
#Then rename the original data set to data.set. After that, we find the tree for the data set
#with the tree() function.
        .First()
        opt.treesize <- 0
        total.error.num <- total.error.denom <- 0
        data.set <- original
        assign("data.set", data.set, frame = 1)
        data.tree <- tree(data.set[, 1] ~ ., data = data.set[, -1])
        assign("data.tree", data.tree, frame = 1)
#The second step: we need an optimal number of terminal nodes for pruning the tree we
#created above. But the problem is we couldn't make sure that the number of terminal
#nodes we obtained from above is optimal. Therefore, we want to find the best size of the
#tree by cross-validating the tree with the function cv.tree(). Then, from this cross-
#validation, which shows the size of the tree, deviance of each number of terminal
#nodes, and other information, we want to find the lowest deviance which we associate
#with the "best" size of the tree. After we obtain the "best" size, we prune the tree. Then,
#we have the lowest deviance and the optimal size of the tree.
#
        data.cv <- cv.tree(data.tree, FUN = prune.tree)
        assign("data.cv", data.cv, frame = 1)
        opt.treesize <- data.cv$size[order(data.cv$dev)[1]]
        assign("opt.treesize", opt.treesize, frame = 1)
        cat("     optimal tree size is ", opt.treesize, 1, "\n")
        cat("     ---- But we'll use ", opt.treesize, 1, " ---\n")
        data.prune <- prune.tree(data.tree, best = opt.treesize, 1)
        assign("data.prune", data.prune, frame = 1)
#
#The third step: before we are ready for running the nnet method, we need to have two
#pieces of information on hand. One is to have the sub-data sets from each terminal node.
#When we prune the tree, we get the best number of the terminal nodes and the sub-data
#sets come with each node by finding the "leaf" of the pruned tree. The other is to have
#the optimal number of terminal nodes or leaves. The optimal number of leaves should
#be equal to the "best" size of the tree we have found by cross-validating. We also could
#double check by looking at the length of the optimal number of leaves.


                data.leaf <-

as.numeric(dimnames(data.prune$frame)[[1]][data.prune$frame[, "var"] == "<leaf>"])

        k.leaves <- length(data.leaf)
        cat("   The value of k.leaves is : ", "\n")
#
#The fourth step: save the unique response values. In neural networks, the nnet function
#treats the data set with 2 levels of response slightly differently from others. Therefore,
#we distinguish the 2-level response data set from other level response data set below
#when running nnet function.
#
        resp.levels <- levels(data.set[, 1])
#
#The fifth step: this step is the most important step in this algorithm. Here, we want to
#conduct neural networks within each leaf of the tree. Since we have the optimal number
#of leaves of the tree, we want to nnet in the leaf one after another. Therefore, first, we
```

```
#locate each node and its own data set (sub-data set of the original). For each leaf, we
#want to find out the error rate of its own by selecting certain different numbers of nodes
#in hidden layer and certain different numbers of random seeds. To stabilize the
#observations of the sub-data set, we scale each sub-data set before they run through nnet
#function. For each for-loop, a detailed description follows.
#
        for(k in 1:k.leaves) {
                sub.leaf.k <- data.set[identify(data.prune, data.leaf[k]), ]
                if(do.scale == T) {
                        num.cols <- sapply(sub.leaf.k, is.numeric)
                sub.leaf.k[, num.cols] <- as.data.frame(scale(sub.leaf.k[, num.cols]))

                }
                assign("sub.leaf.k", sub.leaf.k, frame = 1)
                min.error.rate <- 1.1
#
#In neural networks, the number of nodes in the hidden layer needs to be assigned.
#Different assigned number of the nodes could yield different results along with the
#number of random seed. A number of trials of the combination of a single number of
#nodes in the hidden layer and a single number of random seeds could be tedious as well
#as time-consuming. Therefore, for-loops for the numbers of nodes in the hidden layer
#and the numbers of random seeds could be more effective and could always obtain the
#best results. The worst case complexity would be (k*i*j) in this algorithm.
#
#
#Within the for-loop of random seeds, we sample each sub-data set as training set, and
#the rest is test set. Recall that nnet function treats the number of levels of a data response
#slightly differently. So we created another nnet function called nnet.formula.new for the
#purpose of dealing with 2-class response when we regard it as 2-level response data set.
#Otherwise, we would treat it as multi-class response data set and apply the nnet
#function, when the use.new.version method is false.
#
#
#Within nnet or nnet.formula.new method, the sub-data set goes through the process one
#time with one single number of nodes in the hidden layer and a single number of
#random seeds. The one-time process continues.
#
                for(i in n) {
                        for(j in seed) {
                                q <- j
                                set.seed(q)
                                assign("q", q, frame = 1)
                               sub.samp <- sample(nrow(sub.leaf.k), size =
                               round(nrow(sub.leaf.k) * 0.67, 1), replace = F)
                                assign("sub.samp", sub.samp, frame = 1)
                                if(use.new.version == F)
```

```
                    out <- nnet(sub.leaf.k[, 1] ~ ., data = sub.leaf.k[, -1],
                         size = i, decay =  decay, maxit = 1000, subset =
                         sub.samp, trace = F)
               else out <- nnet.formula.new(sub.leaf.k[, 1] ~ ., data =
                    sub.leaf.k[, -1], size = i, decay = decay, maxit = 1000,
                    subset = sub.samp, trace = F)
               assign("out", out, frame = 1)
#
#When the one-time process comes to the prediction part, it should always have had an
#nnet or nnet.formula.new output. The one-time process checks whether the sub-data set
#has a 2-class response. If it does, the prediction method predicts the results of the test set
#with the output of nnet function. Then it checks its probability with a threshold
#argument, which is usually set to be 0.5. If not, the prediction method will predict the
#probability of data points of each column with the output from the training set and test
#set and then pick up the columns that contain the maximum probability compared with
#others in the same rows. After that, we give the prediction results.
#
               if(use.new.version == F && length(levels(sub.leaf.k[, 1]))
                    == 2) {
                    pred <- predict(out, sub.leaf.k[ - sub.samp,  ])
                    pred <- pred > threshold
                    assign("pred", pred, frame = 1)
               }
               else {
                    pred.mat <- predict(out, sub.leaf.k[ - sub.samp,  ])
                    pred <- dimnames(pred.mat)[[2]][max.col(pred.mat)]
                    assign("pred", pred, frame = 1)
                    pred <- factor(pred, levels = resp.levels)
               }
#
#
#Now that we have the outputs from the nnet or nnet.formula.new function and their
#prediction results, we want to form a table to see the classification accuracy of the one-
#time process. By looking at the results on this table, we want to calculate the
#misclassification error rate. We know that those numbers on the diagonal represent the
#correct classifications. The rest are errors.
#
#
#For a single leaf, we simply sum up all numbers as a total, and sum up the numbers on
#the diagonal as an accurate number. Then, the difference of the total and the accurate
#number is divided by the total and the result is the error rate of that single leaf.
#
#
#For a whole data set, what we are doing is a little different. First of all, we have to
#determine the minimum error rate in the single leaf. Within a single leaf, we have i*j
#one-time processes. The one error rate we prefer is the smallest. After collecting a
```

```
#number of minimum error rates from different leaves, we sum up all of the numbers on
#the diagonal given by those one-time processes that contain the minimum error rates.
#
                                        confuse <- table(sub.leaf.k[ - sub.samp,
                                              dimnames(sub.leaf.k)[[2]][1]], pred)
                                        print(confuse)
                                        assign("confuse", confuse, frame = 1)
                                        sum.confuse <- sum(confuse)
                                        assign("sum.confuse", sum.confuse, frame = 1)
                                        error.rate <- (sum.confuse -
                                          sum(diag(confuse)))/sum.confuse
                                        error.num <- (sum.confuse - sum(diag(confuse)))
                                        error.denom <- sum.confuse
                                        assign("error.rate", error.rate, frame = 1)
                                        if(min.error.rate > error.rate) {
                                         min.error.rate <- error.rate
                                         min.error.num <- error.num
                                         min.error.denom <- error.denom
                                        }
                              }
                    }
                    cat("\n    Giving minimum error rate in the node :  ", round(min.error.rate,
                      3), "\n")
                    cat("\n", "\n")
                    total.error.num <- total.error.num + min.error.num
                    total.error.denom <- total.error.denom + min.error.denom
          }
#
#
#The final step: report the results. We would like to compare the results we obtain from
#this algorithm with other results gathered by previous works. Here we want to report:
#  1.) The optimal size of tree
#  2.) The random seeds we assign to the data set
#  3.) The summary of data classification tree
#  4.) The total number of numerator errors
#  5.) The total number of denominator errors
#  6.) The misclassification error rate of the data
#
#
          cat("****************************************************** ", "\n")
          cat("   ** THE RESULTS OF NEURAL NETWORKS WITHIN
               CLASSIFICATION TREE SIMULATION **",  , "\n")
          cat("\n    optimal tree size is : ", opt.treesize, "\n")
          cat("\n    The number of nodes in the hidden layer is :   ", n, "\n")
          cat("\n    The number of seed is :   ", seed, "\n")
          cat("\n    The summary of this data classification tree is: ", "\n")
```

```
        print(summary(data.prune))
        cat("\n    The simulation produces a total no. of errors and a mean error rate:",
            "\n")
        cat("\n    Giving total no. of numerator errors : ", total.error.num, "\n")
        cat("\n    Giving total no. of denominator errors : ", total.error.denom, "\n")
        mean.error.rate <- total.error.num/total.error.denom
        assign("mean.error.rate", mean.error.rate, frame = 1)
        cat("\n    Giving mean error rate :  ", round(mean.error.rate, 3), "\n")
}
```

C.   whole.tree Algorithm

whole.tree

```
whole.tree <- function(original)
{
# By LTC Chen, Chia-sheng
#
# whole.tree: The purposes of this algorithm are to obtain an original tree of the original
#data, a pruning tree, a cross-validation plot, the optimal size of the terminal nodes, and
#the misclassification error rate of the tree. The outputs of these algorithm serve as basic
#inputs for nnet.in.leaf algorithm.
#
# Arguments:
#       original: full set of data
#
#  return results and value :
#                   1.) Original data tree
#                   2.) Cross-validation plot
#                   3.) Pruned data tree
#                   4.) Optimal number of terminal nodes in classifications tree
#                   3.) Misclassification error rate of the classification tree
#
#
# The overall strategy of this algorithm consists of several steps: First of all, build a
#classification tree for the original data set. Secondly, cross-validate the tree. Next, find
#the optimal number of terminal nodes and the sub-data. After that, as soon as the
#optimal number of the terminal nodes is found, prune the tree with the optimal number
#of nodes. Then, report the results.
#
#
# Before we put the original data set into this algorithm, we arbitrarily relocate the
#response of the original data set on the first column. Then, we define some variables for
#future use. Define variable opt.treesize for the result of cross-validating the tree we get
#from the original data set, and define variables. After that rename the name of the
#original data set as a general term in this algorithm. Now, we are ready for the tree of
```

```
#the data set. We use the tree() function to find the data tree and use the post.tree()
#function to send the tree graph to H drive in the computer for later use.
#
        opt.treesize <- 0
        data.set <- original
        assign("data.set", data.set, frame = 1)# Create a tree for this data set
        data.tree <- tree(data.set[, 1] ~ ., data = data.set[, -1])
        assign("data.tree", data.tree, frame = 1)
        post.tree(data.tree, file = “H:/OriginalTreeOfTheDataSet.ps”)
#
#
#Then, we need an optimal number of terminal nodes for pruning the tree we
#created above. But the problem is we couldn’t make sure that the number of terminal
#nodes we obtained from above is optimal. Therefore, we want to find the best size of the
#tree by cross-validating the tree with the method cv.tree(). Then, from this cross
#validation, which shows the size of the tree, deviance of each “best” size of terminal
#nodes, and other information, we want to find the lowest deviance with the “best” size
#of the tree. After we obtain the “best” size, we prune the tree. Then, we have the optimal
#size of the terminal nodes with the lowest deviance. After cross-validating the original
#tree, we make a plot to show the cross-validation and a graph for pruning tree.
#
#
        data.cv <- cv.tree(data.tree, FUN = prune.tree)
         plot(data.cv)
        assign("data.cv", data.cv, frame = 1)
        opt.treesize <- data.cv$size[order(data.cv$dev)[1]]
        assign("opt.treesize", opt.treesize, frame = 1)
        cat("    optimal tree size is ", round(opt.treesize, 1), "\n")
        cat("    ---- But we'll use ", round(opt.treesize, 1), " ---\n")
        data.prune <- prune.tree(data.tree, best = round(opt.treesize, 1))
        assign("data.prune", data.prune, frame = 1)
        post.tree(data.prune, file = “H:/PruningTreeOfTheDataSet.ps”)
        summary(data.tree)
        summary(data.prune)
}
#
#
# Finally, we want to report the results on the original tree summary as well as the pruned
#tree summary. These two summaries are shown as a comparison for the differences
#between them. The results of the pruned tree summary will be used as inputs for the
#nnet.in.leaf algorithm.
#
#
```

D. whole.nnet Algorithm

whole.nnet

```
function(original, n = 30, seed)
{
# By LTC Chen, Chia-sheng
#
# whole.nnet: The purposes of this algorithm are to obtain a neural network output plot of
#the original data, and the misclassification error rate of the neural networks. The output
#of this algorithm serves as a final result for nnet.in.leaf algorithm.
#
# Arguments:
#        original: full set of data
#               n: number of nodes in the hidden layer of the neural networkclassifier
#            seed: if supplied, pass to set.seed()
#
#  return results and value :
#                    1.) Neural network plot
#                    2.) Misclassification error rate of the classification tree
#
# The overall strategy of this algorithm consists of several steps: First of all, use the
#original data set to randomly create a training set and a test set. Use the training set to
#run the neural network classifier and hold the output. Next, decide the level of the
#response of the original data set (or training set; either one will do.) After that, use the
#output of the nnet function and the test set to predict the misclassification error rate. The
#last, give the neural network plot and the result of the error rate.
#
#In this neural network classifier algorithm, a different number of nodes in the hidden
#layer is able to obtain different result, and so is the number of random seeds. Therefore,
#we pick up a combination of these two numbers which results in the lowest error rate.
#
#Now when we get started with this algorithm, we have to call the nnet function from S-
#Plus library first. In a pre-built function call ".First()", we specified the call of nnet
#function from the S-Plus library and an options function to extend the object size to a
#bigger range. Then set random seed, if necessary.
#
                  .First()
        if(!missing(seed)) set.seed(seed)
#
#Rename the original data set as a common term for general use. Sample the original data
#and name it as the training set and the rest is the test set. We use the training set to run
#the neural networks and save the output. In the nnet function, we set decay to 5e-4, the
#parameter for weight decay to 0.0005; if not specified, default equal to 0.;set maxit =
#1000:  the maximum number of iterations equal to 1000, if not specified, default is 100.
#
#
```

```
        data.set <- original
        assign("data.set", data.set, frame = 1)
        sub.samp.1 <- sample(nrow(data.set), size = round(nrow(data.set) * 0.67, 1),
                replace = F)
        assign("sub.samp.1", sub.samp.1, frame = 1)
        cat("nnet(data.set[,1] ~ ., data = data.set[, -1], size = n, decay = 0.005, maxit =
                1000, subset = sub.samp.1)\n")
        NeuralNetworkOutput <- nnet(data.set[, 1] ~ ., data = data.set[, -1], size = n,
                decay = 0.005, maxit = 1000, subset = sub.samp.1)
        assign("NeuralNetworkOutput ", NeuralNetworkOutput, frame = 1)
        plot(NeuralNetworkOutput)
        cat("\nNow predicting: confusion matrix is\n")
#
#
#Do prediction on the test set, then find the number of the response. If the response has 2
#classes, we set a threshold, 0.5, to the output of the prediction. If the response has more
#than 2 classes, we take the maximum number out of the prediction matrix by rows and
#columns. Then we set a matrix table with the numbers correctly predicted displayed on
#the diagonal and the error numbers on the upper and lower triangles.
#
#
        pred <- predict(NeuralNetworkOutput, data.set[ - sub.samp.1,  ])
        if(length(levels(data.set[, 1])) == 2) {
                pred <- pred > 0.5
                print(pred)
        }
        else {
                pred <- max.col(pred)
        }
        confuse <- table(data.set[ - sub.samp.1, dimnames(data.set)[[2]][1]], pred)
        print(confuse) #
#
# To get the error rate, we sum up all numbers on the matrix table as a total, and sum up
#all the numbers on the diagonal as the correct prediction number. Then the total number
#subtracts the correct prediction number to get the error number. After that, we divide the
#error number by the total number, and we have the error rate. And print it out.
#
        sum.confuse <- sum(confuse)
        error.rate <- (sum.confuse - sum(diag(confuse)))/sum.confuse
        cat("\nGiving an error rate of  ", round(error.rate, 3), "\n")
        invisible(return(error.rate))
}
#
# In chapter 3 of this thesis, we use a modification of the whole.nnet function as part of
#the nnet.in.leaf algorithm. The differences are the input arguments and calculation of
#the misclassification error rates. In the whole.nnet algorithm, we use a single number of
```

```
#nodes in the hidden layer and a single number of random seeds. In the nnet.in.leaf
#algorithm, the number of nodes in the hidden layer is set to be a series of numbers and
#runs with a for-loop. Within the for-loop running with a set of numbers, another for-loop
#is set for the random seeds. Therefore, within a single leaf, we have two for-loops to
#search for error rates. From those error rates we pick up the smallest one and save it as
#the error rate for that particular leaf. Next, we collect the misclassification error rates
#from each leaf by this way. After that, sum up the total error numbers and divide the
#total error numbers by the total numbers we collect. Then, the final result is the
#misclassification error rate for this data by using nnet.in.leaf algorithm.
#
```

E. Examples for Applications to Each Algorithm

The code is an example of how to apply the **nnet.in.leaf** method to a data set. There are seven arguments needed to be considered for this algorithm. They are the data set, the number of nodes in the hidden layer, the random seeds, the scaling, the prediction threshold, the weight decay, and the use of different nnet functions determined by whether the response is 2 levels.

```
> nnet.in.leaf (glass.1, n = (1 : 10), seed = ((1:10)*10), scale = T, threshold =
        0.5, decay = 5e-4, use.new.version = T)
```

The following code is an example of how to apply the **whole.tree** method to a data set. The only one argument needed is the data set.

```
> whole.tree (glass.1)
```

The following code is an example of how to apply the **whole.nnet** method to a data set. There are three arguments required for this algorithm. They are the data set, the number of nodes in the hidden layer, and the random seed.

```
> whole.nnet (glass.1, n = 7, seed = 100)
```

THIS PAGE INTENTIONALLY LEFT BLANK.

APPENDIX B. RAW RESULTS

The results below are reported by data sets. Each data set was examined in a raw unscaled form and in a scaled form where each variable was normalized to mean 0 and standard deviation 1. For each data set, the following is a description of what will be reported:

1. Data set statistics including the size of the data set, the number of classes, along with a description of what the class represents, and the number of independent variables.
2. Classification results for each method as follows:
    a. **Misclassification Error Rate.** An error rate reported by the whole.tree, whole.nnet, or nnet.in.leaf method.

    b. **Optimal Number of Terminal Nodes.** A number that represents the "best" size of a tree.

Data set :	Optimal no. of terminal nodes:	Classifiers:	Whole.tree misclassification error rate	Whole.nnet misclassification error rate	Nnet.in.leaf misclassification error rate
Iris.1	4	Misclassification rate (Unscaled)	0.02667	0	0
		Misclassification rate (Scaled)	0.02667	0	0
Wine.1	7	Misclassification rate (Unscaled)	0.0169	0.017	0.0
		Misclassification rate (Scaled)	0.0169	0.017	0.0
Glass.1	6	Misclassification rate (Unscaled)	0.0	0.0	0.0
		Misclassification rate (Scaled)	0.0	0.0	0.0
Vowel.1	43	Misclassification rate (Unscaled)	0.2121	0.135	0.014
		Misclassification rate (Scaled)	0.2121	0.135	0.023
Letter.1	66	Misclassification rate (Unscaled)	0.3829	0.21	0.055
		Misclassification rate (Scaled)	0.3829	0.21	0.055
Sonar.1	2	Misclassification rate (Unscaled)	0.2404	0.159	0.101
		Misclassification rate (Scaled)	0.2404	0.159	0.101
Diabetes.1	7	Misclassification rate (Unscaled)	0.2279	0.161	0.145
		Misclassification rate (Scaled)	0.2279	0.161	0.125

Table 10. Raw Results for Seven Data Sets

APPENDIX C. WHOLE.NNET METHOD RESULT EXAMPLES

A. EXAMPLES RESULTED FROM WHOLE.NNET METHOD WITH FOLLOWING DATA SETS

These examples are the misclassification results of several data sets using the whole.nnet classifier. The first argument is the name of a data set, the second is the number of nodes in the hidden layer, and the third argument is the random seed, which we set to 100.

The result is a form of a square matrix table. The numbers on the diagonal of the matrix are those correct numbers of the classification. The numbers other than on the diagonal are misclassified numbers. The first column and the first row of the square matrix table are listed either as ordered numbers or as words representing the classes of that data.

The last line of the example is the given error rate of the data set in terms of the given number of nodes in the hidden layer and the random seed.

a. whole.nnet(iris.1, 3, 100)

```
            1  2  3
    Setosa 17  0  0
Versicolor  0 16  0
 Virginica  0  0 17

Giving an error rate of   0
```

This example is the result of the iris.1 data. There are three classes: Setosa, Versicolor, and Virginica. The number of nodes in the hidden layer is three, and the random seed is 100. Clearly, the whole.nnet method gives a misclassification error rate of 0.

b. whole.nnet(wine.1, 3, 100)

```
        Type 1 Type 2 Type 3
 Type 1    20      0      0
 Type 2     0     21      1
 Type 3     0      0     17

Giving an error rate of   0.017
```

This example is the result of the wine.1 data. There are three classes (Type 1, Type 2, and Type 3) which indicate three different classes of wine. The number of nodes in the hidden layer is three, and the random seed is 100. The whole.nnet method gives a misclassification error rate of 0.017.

c. whole.nnet(glass.1, 10, 6000)

```
   1  2 3 4 5 6
1 24  0 0 0 0 0
2  0 24 0 0 0 0
3  0  0 5 0 0 0
5  0  0 0 7 0 0
6  0  0 0 0 3 0
7  0  0 0 0 0 8

Giving an error rate of   0
```

This example is the result of the glass.1 data. There are seven classes (1 through 7) which represent different types of glass. The number of nodes in the hidden layer is ten, and the random seed is 6,000. Obviously, the whole.nnet method gives a misclassification error rate of 0.0.

d. whole.nnet(vowel.1, 10, 1000)

```
     0  1  2  3  4  5  6  7  8  9 10
0   25  3  0  2  0  0  0  0  0  2  0
1    1 25  0  0  4  0  0  0  0  0  0
2    0  0 30  0  0  0  0  0  0  0  0
3    0  1  0 22  4  0  1  0  0  0  0
4    0  0  0  0 26  0  1  0  0  0  0
5    0  0  0  0  0 31  1  2  0  0  0
6    0  0  3  0  1  1 21  0  0  0  0
7    1  0  0  0  0  2  0 26  1  1  0
8    0  0  0  0  0  0  0  2 27  2  2
9    0  2  1  0  0  0  0  0  0 25  2
10   0  0  0  0  0  0  0  0  0  1 25

Giving an error rate of   0.135
```

This example is the result of the vowel.1 data. There are eleven integer classes (0 through 10) which index different vowels. The number of nodes in the hidden layer is ten, and the random seed is 1,000. Obviously, the whole.nnet method gives the misclassification error rate of 0.135.

e. whole.nnet(letter.1, 10, 200)

```
      1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18   19   20   21   22   23   24   25   26
A   221    0    0    2    0    1    2    1    0   13    1    4    1    0    1    0    0    0    2    0    2    1    1    1    6    4
B     0  192    0    6    0    3    4    4    0    0    0    0    0    0    0    2    0   36    4    0    1    1    1    4    2    0
C     0    0  204    0   13    2    8    0    0    0    2    2    0    0    2    0   11    0    0    0    2    0    0    0    0    0
D     1   15    0  196    0    0    0   11    0    2    0    0    2    9    3    1    0   14    5    0    2    0    0    1    1    1
E     0    3    4    0  191    1   11    1    0    0    5    4    0    0    0    0    2    3    5    4    0    0    0    7    0    8
F     0   13    0    1    8  194    4    2    1    3    0    0    0    5    0   12    1    2    3   20    0    0    0    3    2    0
G     1    2   11    2    1    3  164    7    0    1    2    6    0    1    6    4   31    6    7    0    0    0    1    0    0    0
H     1    5    2   11    1    3    1  148    0    0    9    1    0   11    6    0    2   21    0    1    6    1    1    3    1    0
I     1    2    0    5    2    4    0    0  199   10    0    5    0    0    0    2    6    0    6    0    0    0    0    2    2    5
J     3    1    0    2    0    4    0    1    7  205    0    2    0    1    1    2    2    0   12    0    0    0    0   11    0    3
K     0    1    2    3   10    0    2   11    0    0  179    3    0    2    0    0    2   20    0    0    3    0    0   12    1    0
L     2    0    0    1    7    0   10    0    0    5    0  212    0    0    0    0    2    5    2    6    0    0    0   12    3    0
M     2    0    0    0    0    0    2    2    0    0    0    0  260    4    1    0    0    3    0    0    1    0    4    0    0    0
N     3    3    0    3    0    0    0    8    0    0    2    0   10  205    4    0    0    3    0    0    2    0    7    0    0    0
O     1    1    1    4    1    1    5    3    0    2    0    0    1    0  198    2    4    6    1    0    8    0   11    0    0    0
P     0    5    0    0    0   16    5    0    1    0    0    0    0    0   11  212    0    0    0    0    0    2    4    0    4    0
Q     0    5    3    1    1    1    9    0    0    0    0    2    0    0   13    0  214    1    7    1    0    0    0    0    1    2
R     0   11    0   11    0    0    2   11    0    0    9    0    1    1    5    1    0  194    0    0    0    0    0    0    1    0
S     0   12    0    3   14    3    0    1    4    5    1    8    0    0    0    0    4    0  145    2    0    0    0    8    1    8
T     0    2    0    0    7    4    0   11    1    2    2    0    0    0    0    0    1    3    8  195    1    0    0    0   14    5
U     2    1    2    1    0    0    0    2    0    0    2    0    4    6    2    0    2    1    0    0  211    1    9    0    0    0
V     1    1    0    0    0    4    0    1    0    0    0    0    0    0    1    2    2    5    0    0    2  209    9    0    2    0
W     0    0    0    0    0    0    1    0    0    0    0    0    8    1    1    0    0    0    0    0    3    3  246    0    0    0
X     0    4    2    1   10    0    0    3    5    0   11    1    0    0    0    0    6    1   13    3    1    0    0  198    4    2
Y     2    0    0    0    0    4    0    0    0    0    0    1    0    0    0    2    2    0    0   11    3    4    0    3  223    0
Z     0    1    0    0    5    2    0    0    0    7    0    0    0    0    0    0    5    0   17    1    0    0    0    0    0  198

Giving an error rate of   0.21
```

This example is the result of the letter.1 data. There are 26 classes (A through Z) which are the targets to be identified. The number of nodes in the hidden layer is ten, and the random seed is 200. It is obvious that the whole.nnet method gives the misclassification error rate of 0.21.

f. whole.nnet(sonar.1, 10, 200)

```
    FALSE TRUE
M     27    3
R      8   31

Giving an error rate of   0.159
```

This example is the result of the sonar.1 data. There are two classes, representing sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. (M and R). The FALSE and TRUE shows that the threshold is set to 0.5. The number of nodes in the hidden layer is ten, and the random seed is 200. Clearly, the whole.nnet method gives the misclassification error rate of 0.159.

g. whole.nnet(diabetes.1, 7, 7000)

```
   FALSE TRUE
0   148   27
1    14   65

Giving an error rate of   0.161
```

This example is the result of the diabetes.1 data. There are two classes, one representing people having diabetes and zero for those who don't have diabetes. The FALSE and TRUE shows that the threshold is set to 0.5. The number of nodes in the hidden layer is seven, and the random seed is 7000. It is obvious that the whole.nnet method gives the misclassification error rate of 0.161.

B. THE ERROR RATES FOR EXAMPLE OF NEURAL NETWORK CLASSIFICATION IN CHAPTER 2

Misclassification Error Rates Resulted from wine.1 Data Set Using Neural Network Classifier whole.nnet

n	Minimum Error rate	Higher error rate	n	Minimum Error rate	Higher error rate
1		0.11864	16		0.05085
2		0.08475	17		0.0339
3	0.01695		18		0.0339
4	0.01695		19	0.01695	
5		0.05085	20	0.01695	
6		0.05085	21		0.05085
7	0.01695		22		0.0339
8	0.01695		23		0.0339
9	0.01695		24	0.01695	
10	0.01695		25	0.01695	
11	0.01695		26	0.01695	
12		0.0339	27	0.01695	
13		0.05085	28	0.01695	
14	0.01695		29	0.01695	
15		0.05085	30	0.01695	

Table 11: These Error Rates Gathered by Using whole.nnet Method with Random Seed 100

To see the application of the whole.nnet method, we tried different numbers of nodes in the hidden layer between 1 and 30 nodes with random seed 100. In Table 11, the misclassification rates did not remain the same when the random seed was fixed.

C. THE AF.1 DATA SET

The AF.1 data set is collected to estimate the cost of purchasing different types of military aircraft. Based on 920 observations, there are 11 attributes associated with the cost estimation. The response is the aircraft model. The major attributes are source, location, year, cost, fuel, and personnel pay. The response has 50 classes. The number of classes of the response is too large to fit the nnet.in.leaf method. When we applied the data to the tree function, it didn't work, either. The tree function could not handle such a large number of classes as in the response of the AF.1 data.

LIST OF REFERENCES

Aeberhard, S., Coomans D. and de Vel, O. (1992), *Comparison of Classifiers in High Dimensional Settings,* Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of Mathematics and Statistics, James Cook University of North Queensland. (Also submitted to Technometrics).

Breiman, L., Friedman, J. H., Olshen, R. A. and Stone, C. J. (1984), *Classification and Regression Trees*, Belmont, CA: Wadsworth.

Dasarathy, B.V. (1980), "Nosing Around the Neighborhood: A New System Structure and Classification Rule for Recognition in Partially Exposed Environments." IEEE ransactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 1, 67-71.

Devore, J.L.(1995), *Probability and Statistics for Engineering and the Sciences,* 4th ed., Brooks/Cole Publishing Co.

Fisher, R. A.(1936), The Use of Multiple Measurements in Taxonomic Problems, *Annals of Eugenics 7,* 179 - 188.

Frey P. W. and Slate D. J. (1991),"Letter Recognition Using Holland-style Adaptive Classifiers." Machine Learning Vol 6 #2 March 91, UC Irvine Machine Learning Data Repository, Web site: http://www.ics.uci.edu/~mlearn/MLRepository.html

Hebb, D. O. (1949), *The Organization of Behavior.* New York: Wiley.

Hertz, J., Krogh, A., & Palmer, R. G. (1991), *Introduction to the Theory of Neural Computation.* Redwood City, CA: Addison-Wesley.

Hinton, G. E. (1986), Learning Distributed representations of Concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society (Amherst, 1986),* pp. 1-12. Hillsdale: Erlbaum.

Karo, C. (1998), *Two Nearest Neighbor Classification Rules,* Monterey CA: Naval Postgraduate School, Master's thesis.

Kobayashi, Izumi (1999), *Sensitivity Analysis of the Topology of Classification Trees,* Monterey CA: Naval Postgraduate School, Master's thesis.

McCulloch, W. S. & Pitts, W. (1943), A Logical Calculus of Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics 5,* 115 – 133.

Minsky, M. L. & S. A. Papert (1969), *Perceptrons.* Cambridge: MIT Press. Partially reprinted in Anderson and Rosenberg [1988].

Morgan, J. N. & Sonquist, J. A. (1963), Problems in the Analysis of Survey Data, and a Proposal. *Journal of the American Statistical Association 58*, 415 – 434.

Ripley B.D., (1996), *Pattern Recognition and Neural Networks,* Cambridge, UK. Cambridge University Press.

Rohwer R., Wynne-Jones M., & Wysotzki F. (1994), *Neural Networks, Machine Learning, Neural and Statistical Classification,* Ellis Horwood.

Rosenblatt, F. (1962), *Principles of Neurodynamics*. New York: Spartan.

S-PLUS 2000, (1999), Guide to Statistics, Data Analysis Products Division, MathSoft, Seattle, WA.

StatLib - Software and Extensions for the S (S-Plus) language [http://lib.stat.cmu.edu/S/] (for down loaded S-Plus classification library).

Venables W.N., & Ripley B.D., (1999), *Modern Applied Statistics with S-Plus (Statistics and Computing)*,San Francisco, CA, Springer.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Professor Samuel E. Buttrey 1
   Code OR/SB
   Naval Postgraduate School
   Monterey California 93943-5002

4. Professor Lyn Whitaker.. 1
   Code OR/LA
   Naval Postgraduate School
   Monterey California 93943-5002

5. Director.. 1
   Fu-Shin-Kang College, English Training Center
   Pei-To, Taipei City
   Taiwan, The Republic of China

6. LTC Chen, Chia-sheng.. 6
   29-2F, Shung-Fong Rd.
   Tu-Cheng City (236), Taipei County
   Taiwan, The Republic of China

0052